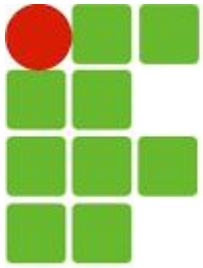


INSTITUTO FEDERAL
GOIÁS
Câmpus Anápolis

Estrutura de Dados II

Sérgio Canuto
Sergio.canuto@ifg.edu.br

Pré-requisitos…

- Ponteiros e alocação dinâmica de memória.
- Tipos abstratos de Dados.
  - Listas e suas generalizações:
    - Tipos, operações e implementação.
  - Pilhas e filas:
    - Tipos e implementação.
- Algoritmos de Pesquisa e Ordenação:
  - Bolha, inserção e seleção, quick sort, Merge Sort, Counting Sort, Radix Sort,Bucket Sort.
- Notação e Análise Assintótica.

Objetivos

- Permitir ao aluno manipular estruturas de dados avançadas, de forma que o mesmo consiga entender o funcionamento de diversas estruturas e suas complexidades.
- Capacitar o aluno a analisar e construir estruturas de dados adequadas para aplicações específicas.

Habilidades e Competências

- Resolver problemas usando ambientes de programação.
- Aplicar os princípios de gerência, organização e recuperação da informação.
- Reconhecer a importância do pensamento computacional no cotidiano e sua aplicação em circunstâncias apropriadas e em domínios diversos.

Ementa

- Tabelas Hash.
  - Endereçamento aberto, fechado e misto.
- Estruturas Heap e Heap Sort.
- Árvores:
  - Balanceamento.
  - Árvores rubro-negras.
  - Árvores B.
  - Árvores AVL.
- Equações de recorrência.
- Introdução a compressão de dados.

Ementa

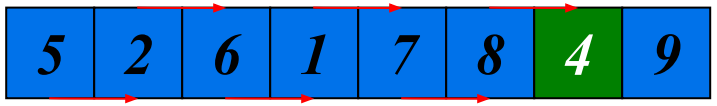
- Tabelas Hash.
  - Endereçamento aberto, fechado e misto.
- Estruturas Heap e Heap Sort.
- Árvores:
  - Balanceamento.
  - Árvores rubro-negras.
  - Árvores B.
  - Árvores AVL.
- Equações de recorrência.
- Introdução a compressão de dados.

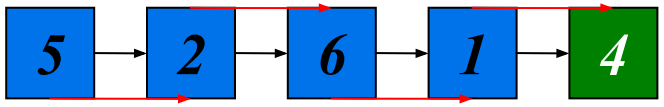
Tabelas Hash

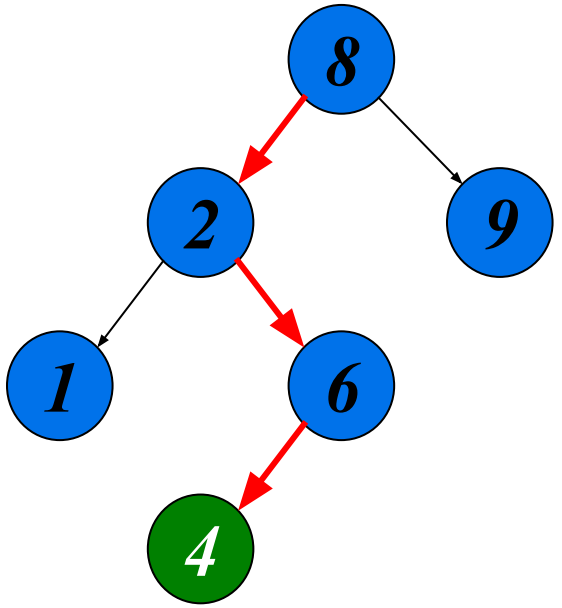
Tabelas Hash - *Motivação*

- *Procurar, inserir e apagar registros rapidamente baseados nas suas chaves*
- *Estruturas de busca sequencial/binária levam tempo até encontrar o elemento desejado.*

Ex: Arrays e listas *Ex: Árvores*







Tabelas Hash - *Motivação*

Suponha que você pudesse criar um array onde qualquer item pudesse ser localizado através de acesso direto.

Isso seria ideal em aplicações do tipo Dicionário, onde gostaríamos de fazer consultas aos elementos da tabela em tempo constante.

Tabelas Hash - *Motivação*

•*Busca de elementos em base de dados: estruturas de dados em memória, bancos de dados e mecanismos de busca na internet.*

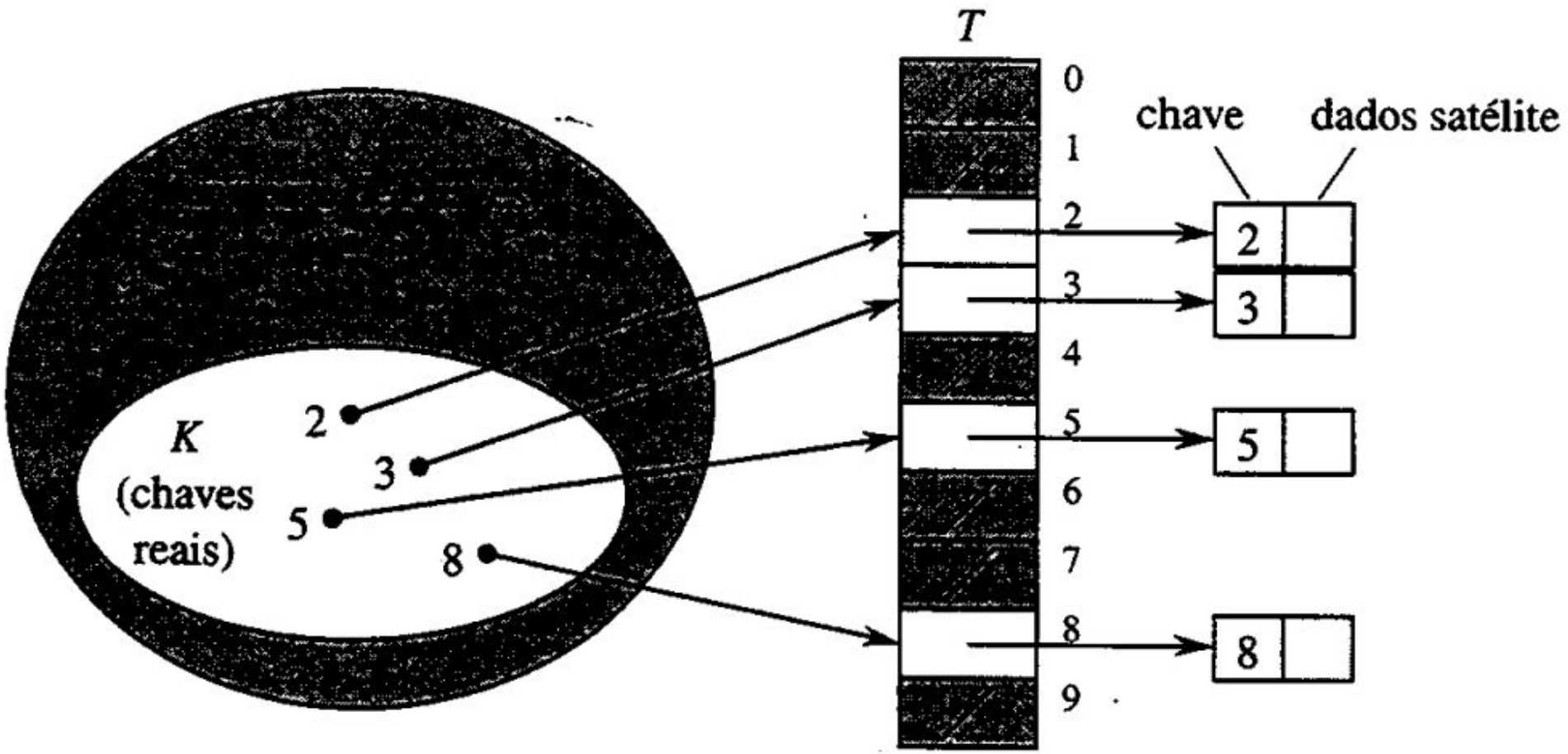
•*Verificação de integridade de dados e autenticação de mensagens: os dados são enviados juntamente com o resultado da função de hashing. Quem receber os dados recalcula a função de hashing usando os dados recebidos e compara o resultado obtido com o que recebeu. Se os resultados forem diferentes, houve erro de transmissão.*

•*Armazenamento de senhas com segurança: a senha não é armazenada no servidor, mas sim o resultado da função de hashing.*

•*Implementação da tabela de símbolos dos compiladores.*

•*Criptografia: MD5 e família SHA (Secure Hash Algorithm).*

Tabela de Endereçamento Direto



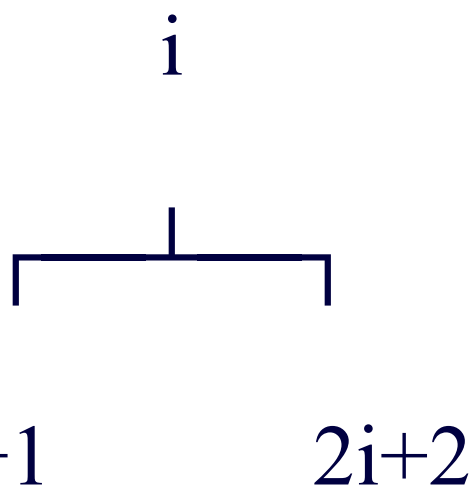
Heap

Heap

- “Árvore binária armazenada em um vetor”

Suponha dado um vetor A[0..n]. Por enquanto, os valores armazenados no vetor não nos interessam; só interessam certas relações entre índices. Para todo índice i, diremos que

- 2i+1 é o filho esquerdo de i,
- 2i+2 é o filho direito de i



Heap

- “Árvore binária armazenada em um vetor”

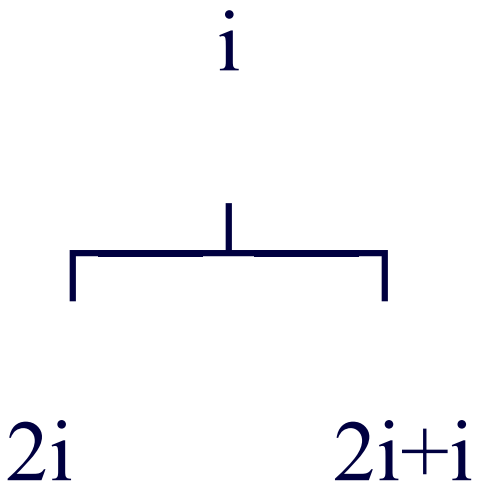
Suponha dado um vetor A[1..n] (e não A[0…n], como vimos antes)... Como seriam as regras?

Heap

- “Árvore binária armazenada em um vetor”

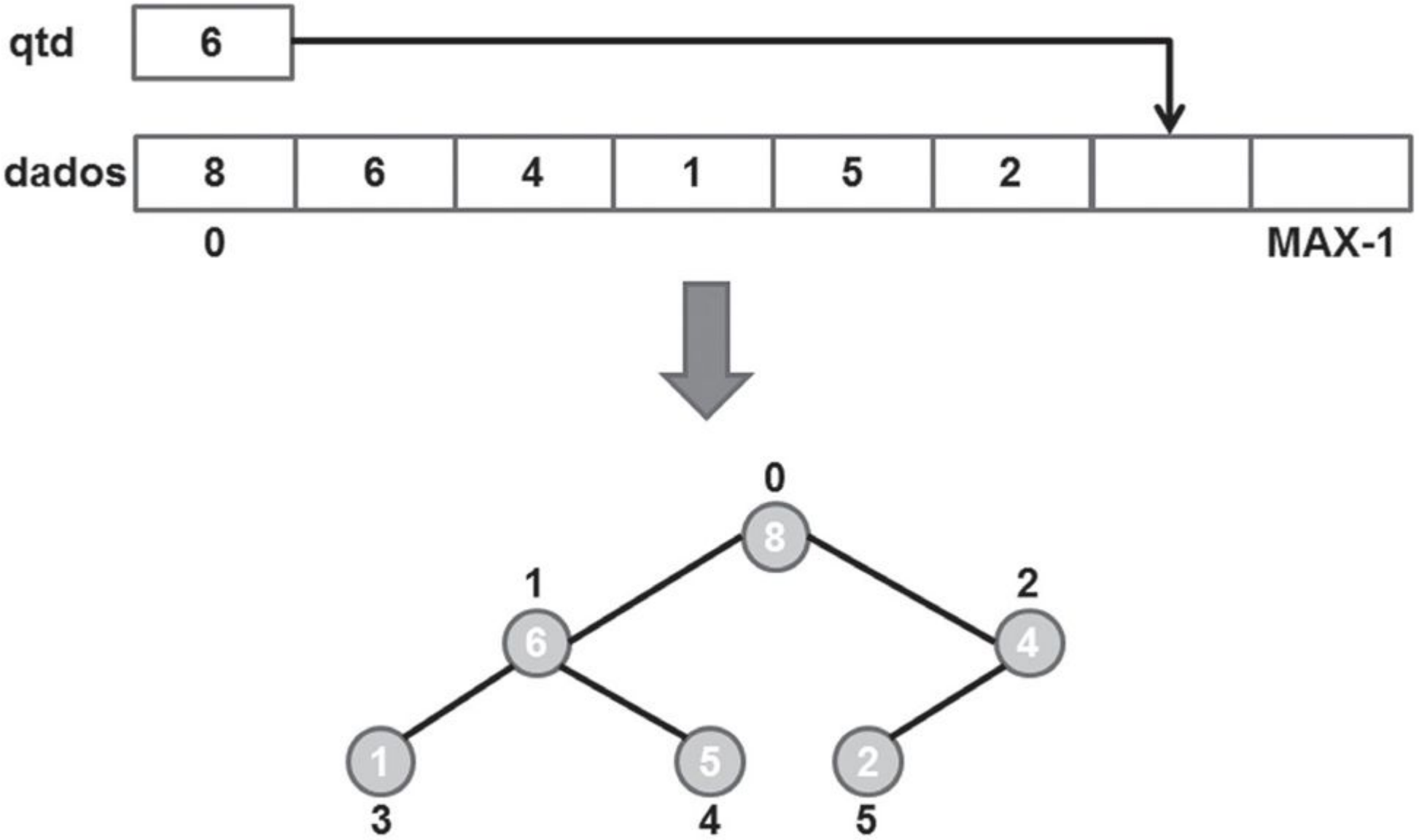
Suponha dado um vetor A[1..n] (e não A[0…n])... Como seriam as regras?

- 2i é o filho esquerdo de i,
- 2i+1 é o filho direito de i



Heap

- “Árvore binária armazenada em um vetor”



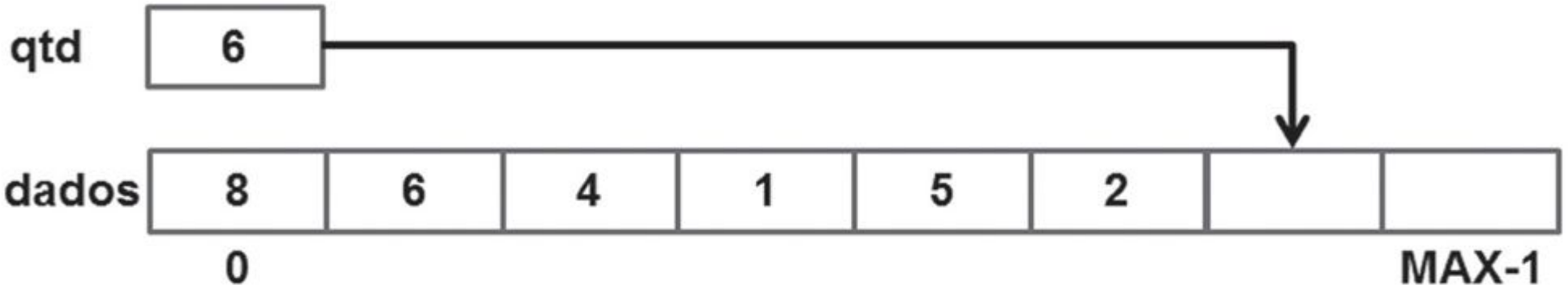
Heap

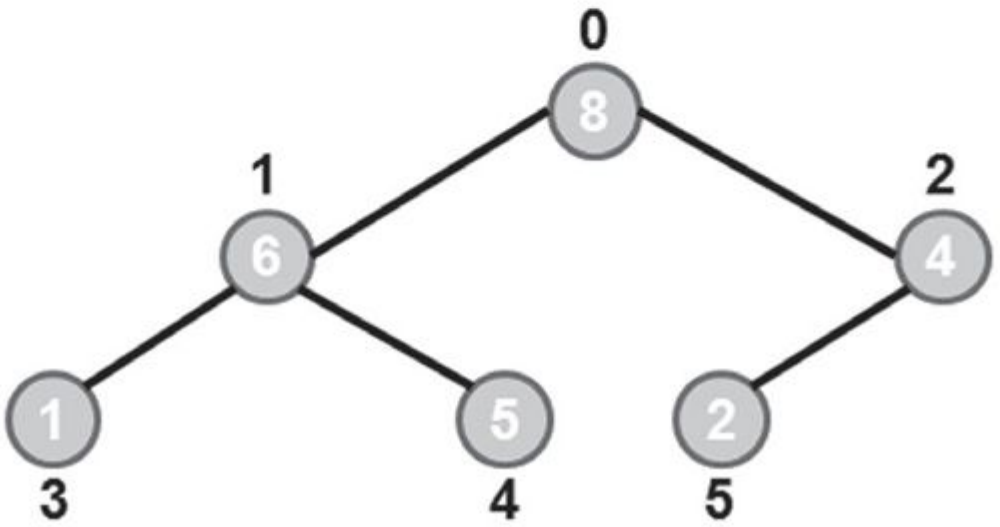
- “Árvore binária armazenada em um vetor”

- Caso de Uso: Fila de Prioridades
- Regras:
  - O conteúdo do nó pai tem sempre uma prioridade maior ou igual à prioridade de seus filhos.
  - Desse modo, o elemento de maior prioridade estará sempre no início do array (início da fila)

Heap

- “Árvore binária armazenada em um vetor”





8 > 6 e 8>4

6>1 e 6>5; 4>2

Heap

- “Árvore binária armazenada em um vetor”
- Caso de Uso: Fila de Prioridades
  - Neste tipo de fila, os elementos nela inseridos possuem um dado extra, associado a eles: a sua prioridade.
    - É o valor associado à prioridade que determina a posição de um elemento no início da fila

Heap

- Ex. de usos da Fila de Prioridade:

•Uma fila de pacientes esperando transplante de fígado.

•Busca em grafos (algoritmo de Dijkstra).

•Compressão de dados (código de Huffman).

•Sistemas operacionais (manipulação de interrupções).

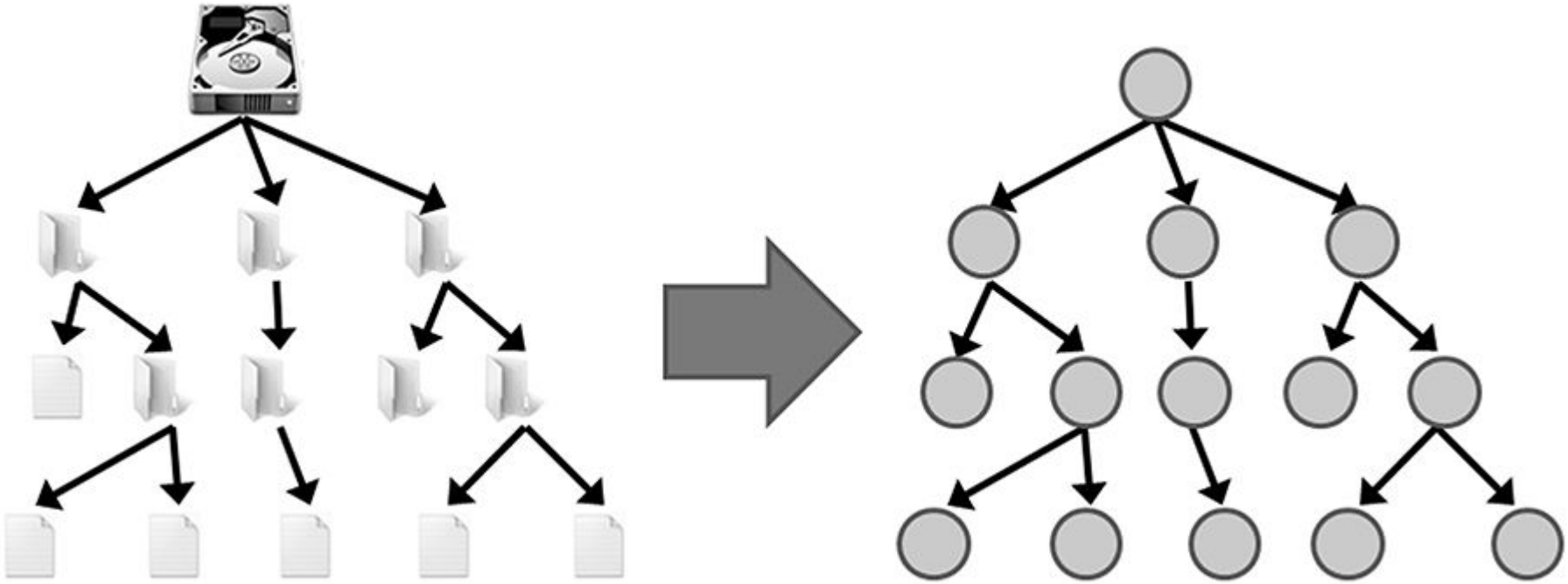
•Fila de pouso de aviões em um aeroporto (prioridade por combustível disponível).

Heap

- “Árvore binária armazenada em um vetor”
- Caso de Uso: Ordenação
  - Garantindo que o nó pai é sempre maior que os demais:
    - Podemos, a cada iteração, remover o nó pai do heap e inseri-lo em um vetor ordenado.

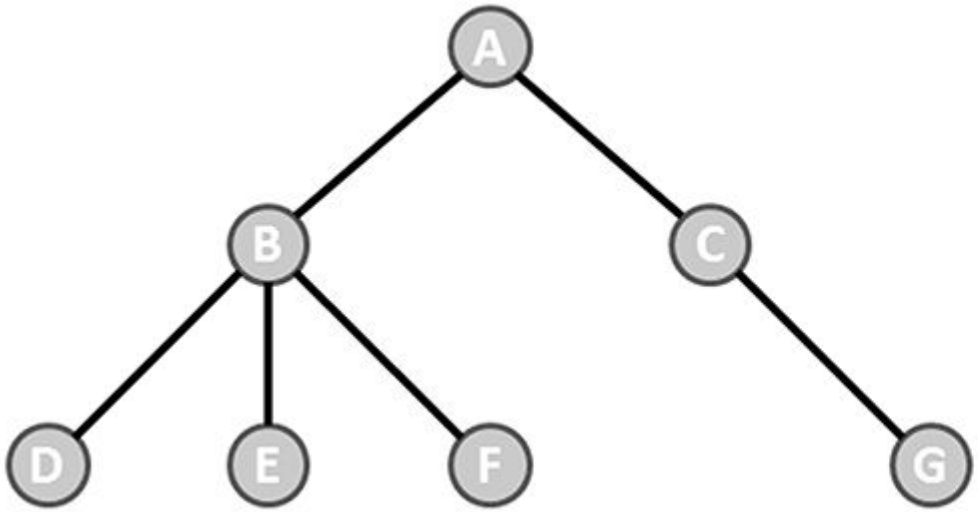
Árvores

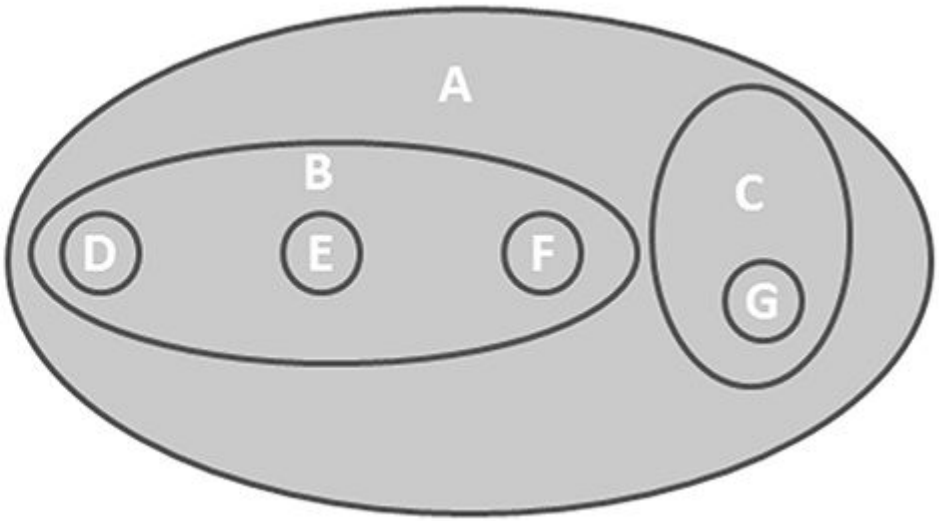
- Diversas aplicações necessitam que se represente um conjunto de objetos e as suas relações hierárquicas.



Árvores

- Diversas aplicações necessitam que se represente um conjunto de objetos e as suas relações hierárquicas.



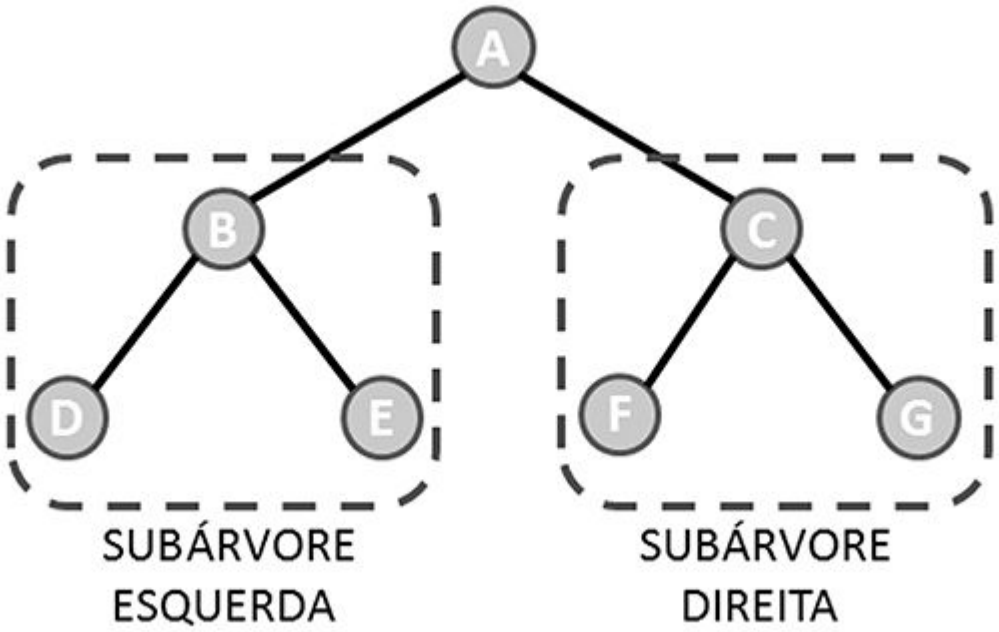


Árvores

- Diversas aplicações necessitam que se represente um conjunto de objetos e as suas relações hierárquicas.
  - Relações de descendência (pai, filho etc.).
  - Diagrama hierárquico de uma organização.
  - Campeonatos de modalidades desportivas.
  - Taxonomia.
  - Busca de dados armazenados no computador.
  - Representação de espaço de soluções (ex: jogo de xadrez).

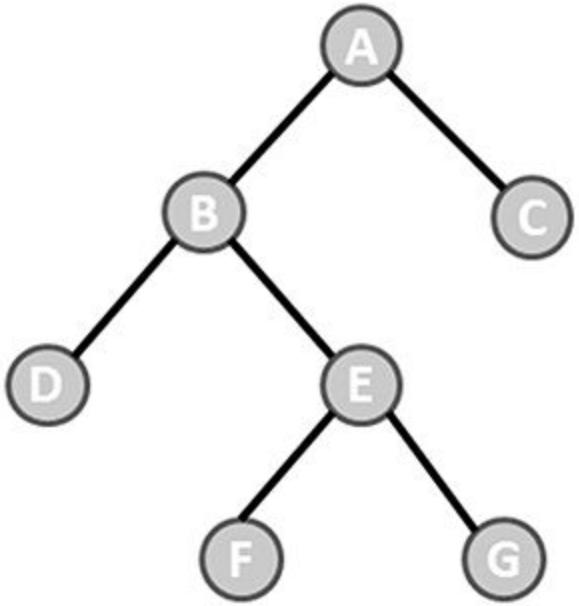
Árvores

- Árvores Binárias

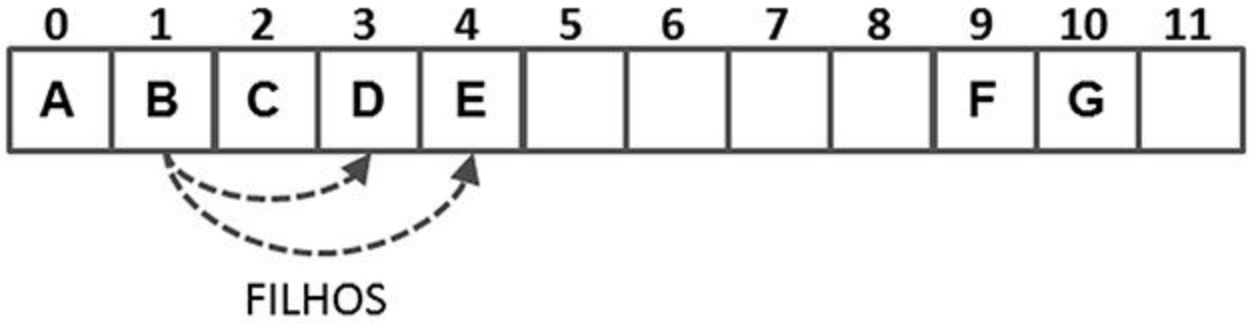


Árvores

- Árvores Binárias (heap)

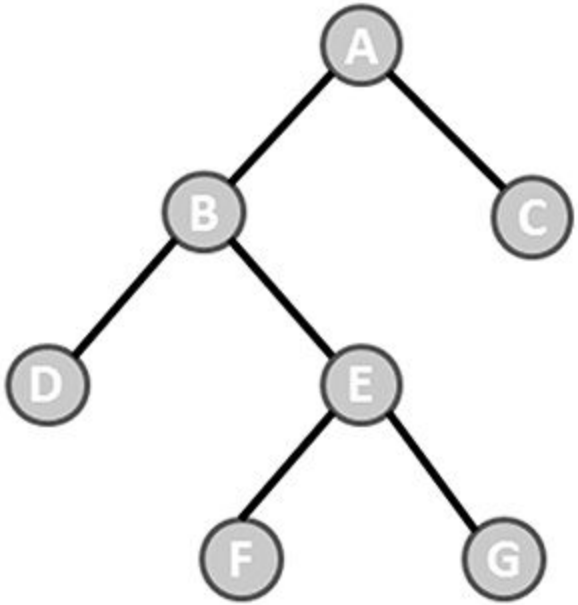


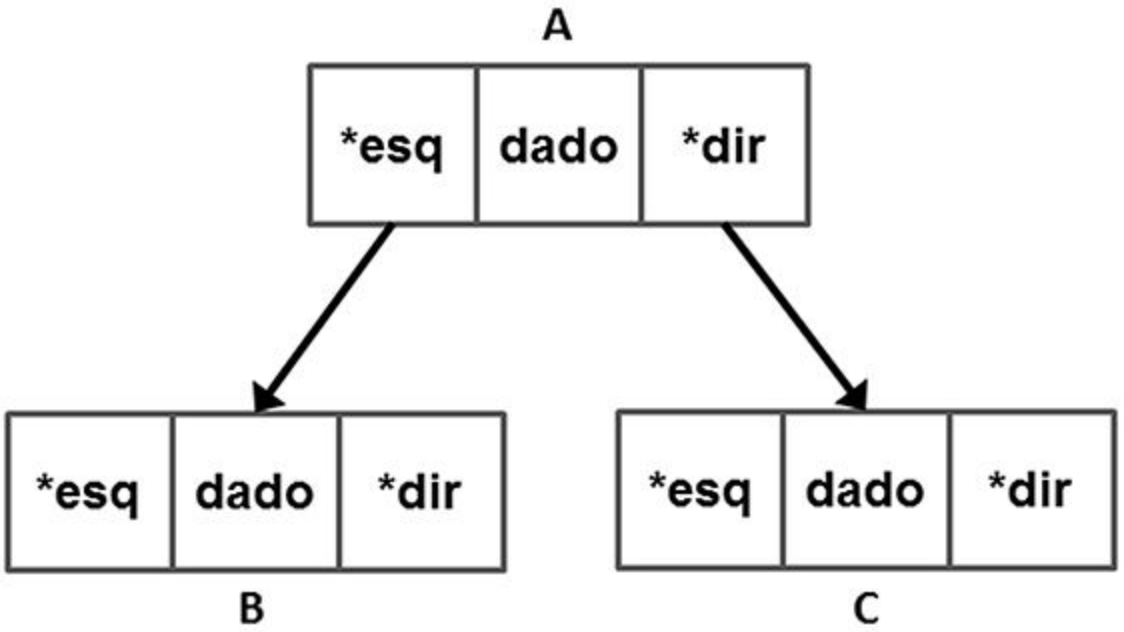
FILHO_ESQ(PAI) = 2 * PAI + 1
FILHO_DIR(PAI) = 2 * PAI + 2



Árvores

- Árvores Binárias (acesso encadeado)



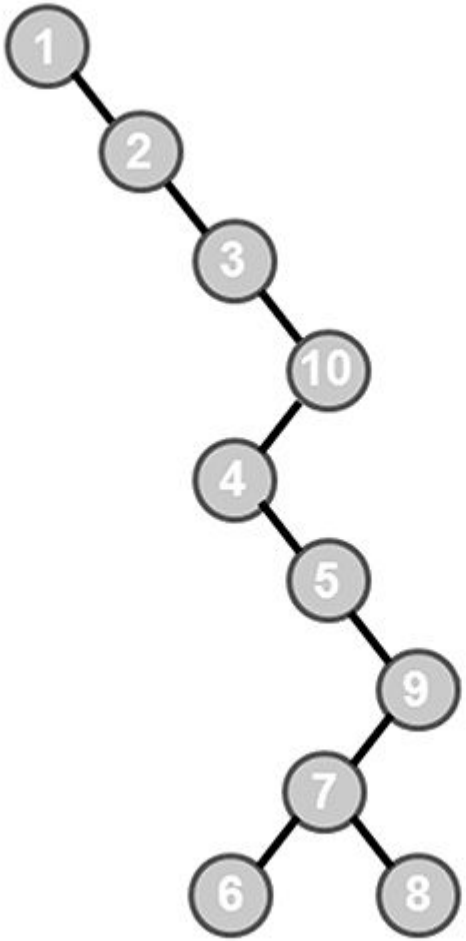


Árvores

- Árvores Binárias (operações)
    - Criação da árvore.
    - Inserção de um elemento.
    - Remoção de um elemento.
    - Busca por um elemento.
    - Destruição da árvore.
    - Além de informações com número de nós, altura ou se está vazia.

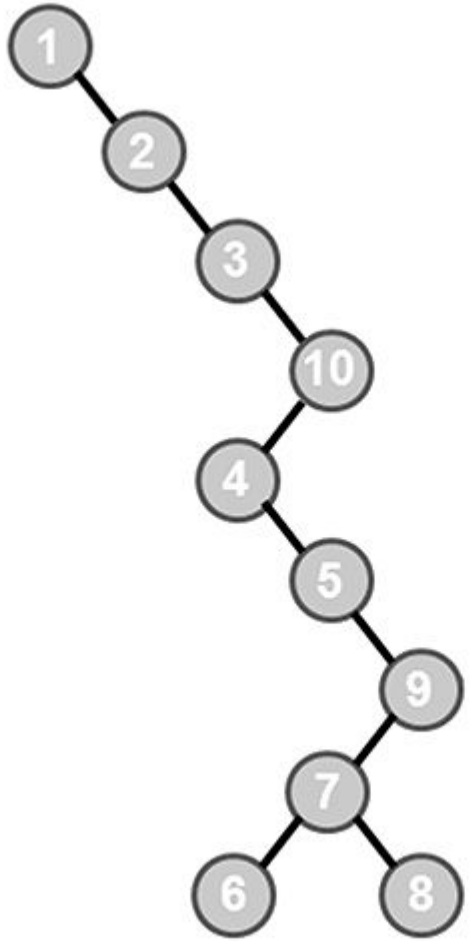
Árvores

- Árvores Binárias (balanceamento)
    - E se tivéssemos uma árvore assim?



Árvores

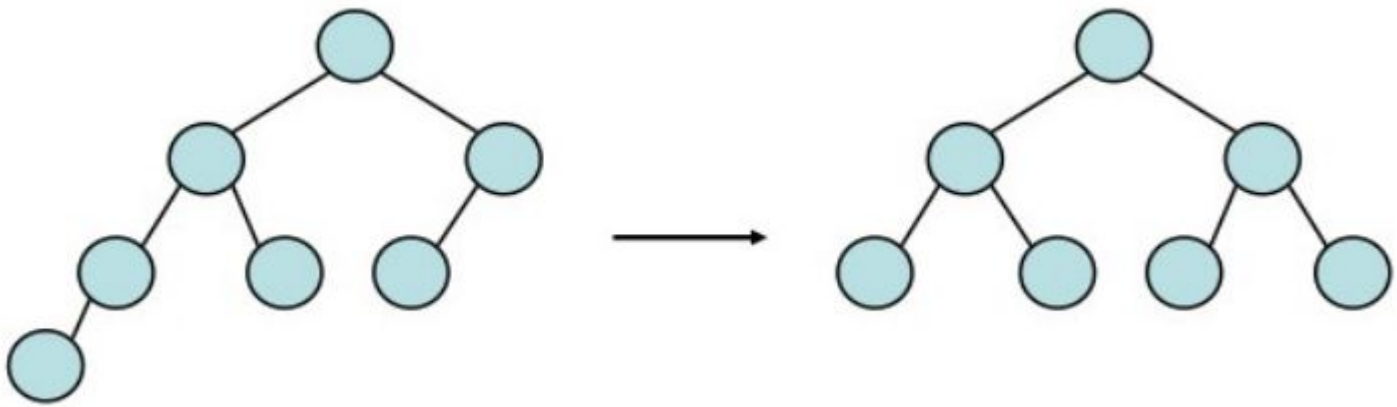
- Árvores Binárias (balanceamento)
    - E se tivéssemos uma árvore assim?
        - Precisaríamos balanceá-la!



Árvores

Embora uma árvore completa possua altura proporcional a *log n*, após apenas uma operação de inserção, a árvore pode perder essa característica.

Uma solução seria aplicar um algoritmo que tornasse a árvore novamente completa, porém, o custo dessa operação seria no mínimo proporcional a n, ou seja, Ω(n).



Árvores

- Árvores Binárias (balanceamento)
  - Solução para inserções/remoções:
    - Árvore AVL.
    - Árvore Red-Black (rubro-negra).
    - Árvore B, B+ e B*.

Árvores

Árvores (balanceamento)

- Objetivo: manter o custo das operações na mesma ordem de grandeza da altura de uma árvore completa, ou seja, O(log n), onde n é o número de nós da árvore.
- Árvore balanceada: o custo das operações de busca, inserção, remoção e arrumação da árvore mantém-se em O(log n).
- Há implementações de árvores de busca binária balanceadas que garantem altura O(log n), tais como árvores rubro-negras e árvores AVL.
- As árvores B também são consideradas balanceadas, mas não são binárias e possuem como característica o armazenamento de mais de uma chave por nó.

Recorrência

Recorrência

Descrevendo o tempo de execução do problema:

```
public int fatorial(int n) {
    if (n==0 || n == 1)
        return 1;
   else
       return n * fatorial(n-1);
}
```

Recorrência

Descrevendo o tempo de execução do problema:

```
public int fatorial(int n) {
    if (n==0 || n == 1)
        return 1;
   else
        return n * fatorial(n-1);
}
```

… com “Relações de Recorrência”!

- Relação de recorrência é uma equação ou inequação que descreve uma função em termos dela mesma considerando entradas menores.

Recorrência

Descrevendo o tempo de execução do problema:

```
public int fatorial(int n) {
    if (n==0 || n == 1)
        return 1;
   else
        return n * fatorial(n-1);
}
```

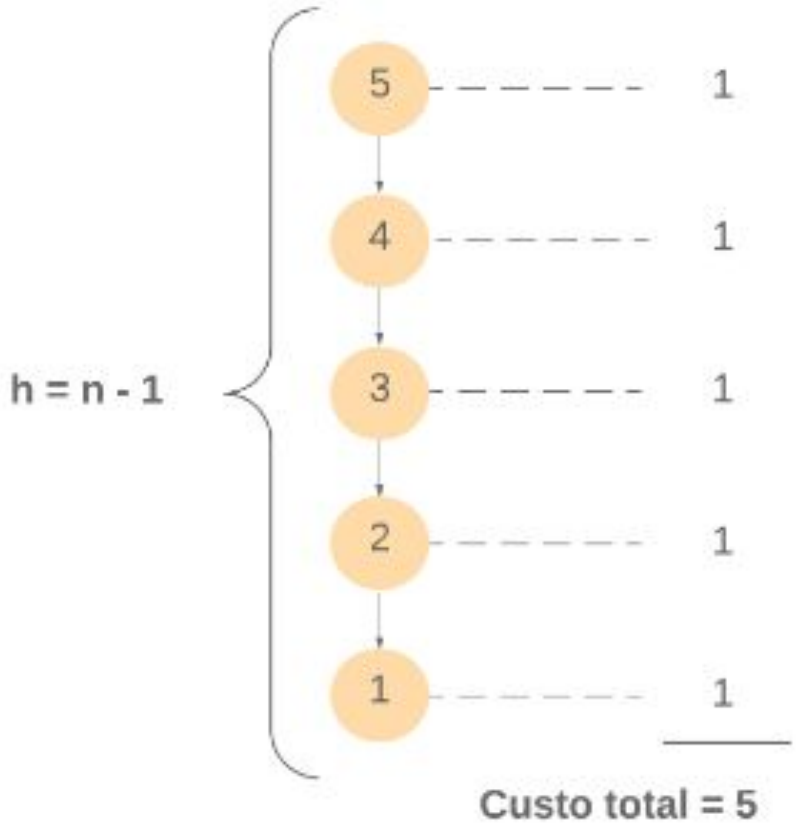
… com “Relações de Recorrência”!

- A função que descreve o tempo de execução de um algoritmo recursivo é dada por sua relação de recorrência. No algoritmo de cálculo do fatorial:

- $T(n)=T(n-1)+\Theta(1)$

Recorrência

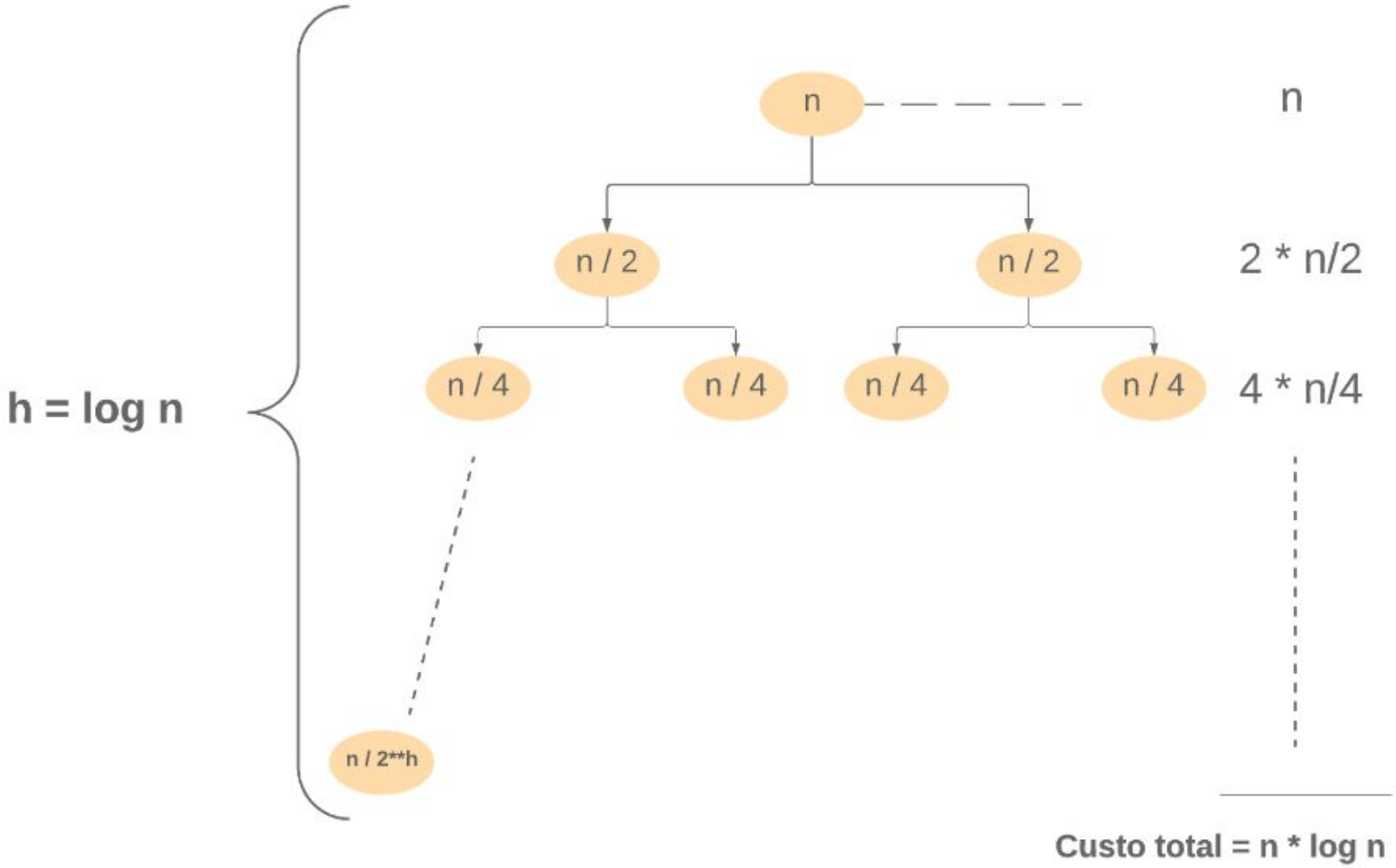
- $T(n)=T(n-1)+\Theta(1)$
- Analisando com o método da árvore de recursão:



Recorrência

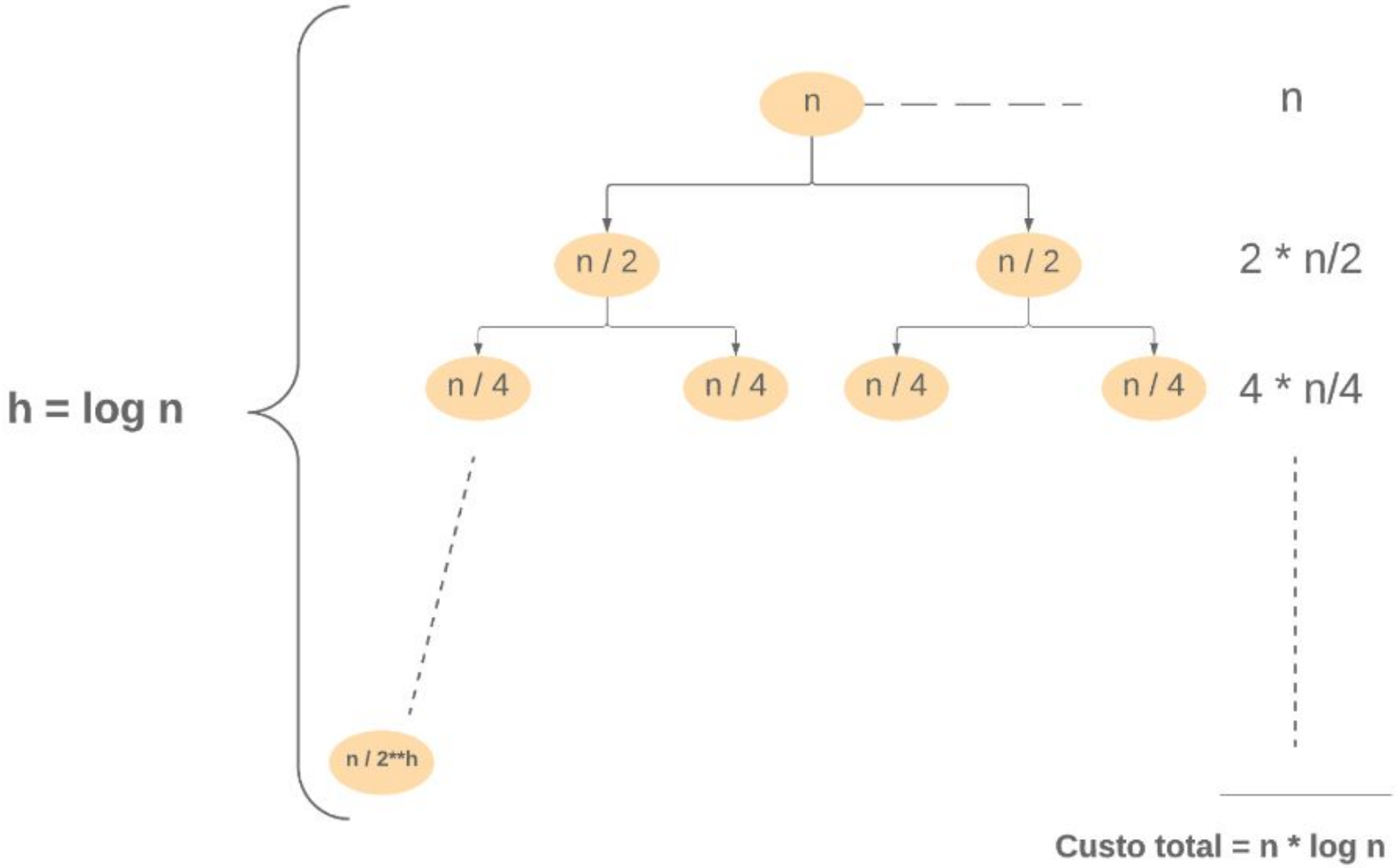
- $T(n)=2T(n/2)+n$

- Analisando com o método da árvore de recursão:



Recorrência

- Outras formas de Análise:
  - Método da Substituição
  - Método Mestre



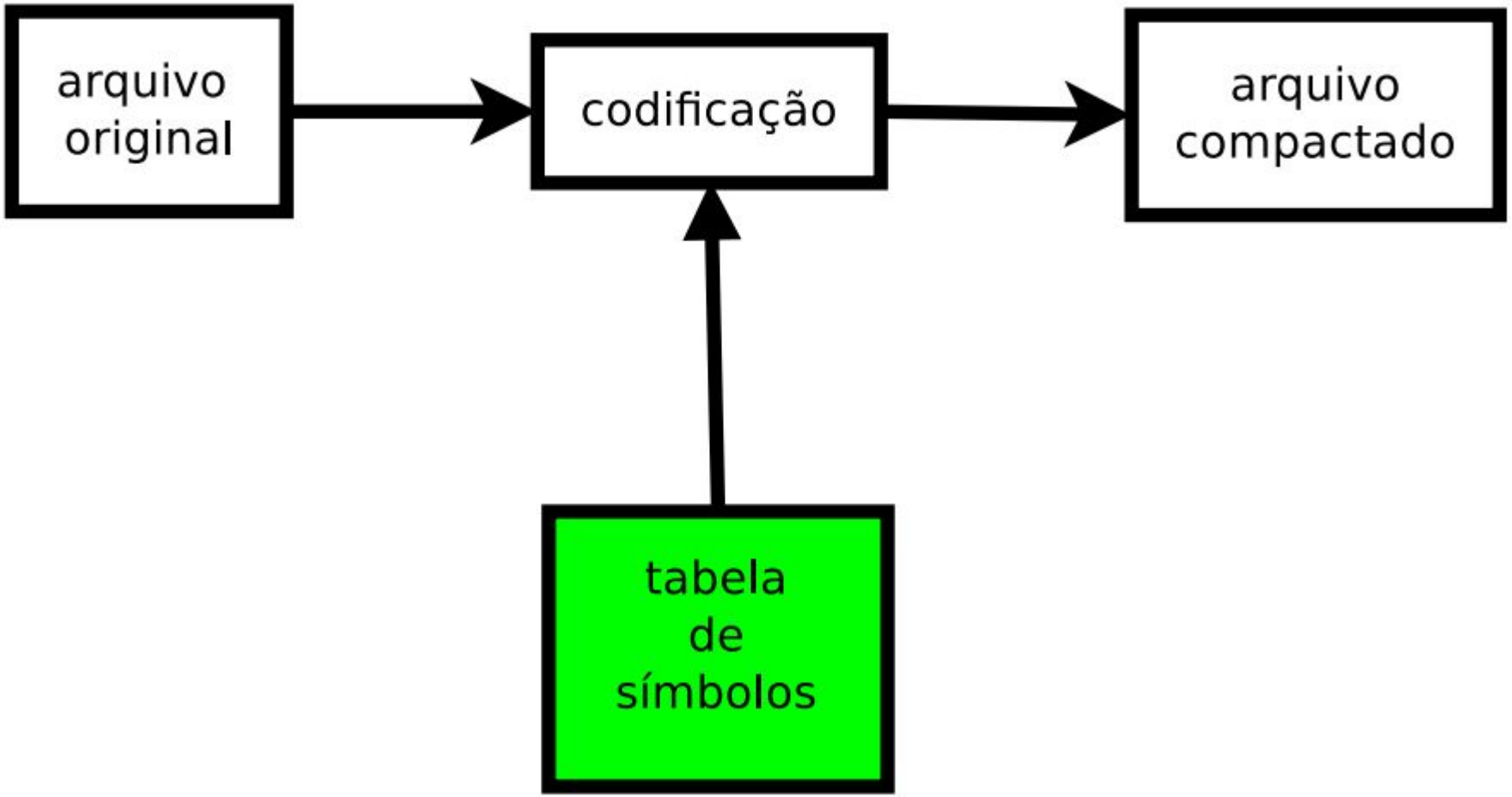
Introdução à Compressão de Dados

- Conteúdo do arquivo:
- AAEIIIIIOOOOOUAAAAAAAAAE
- Tamanho em disco: 24 bytes Tabela de símbolos - ASCII

carácter	decimal	binário (8bits)
A	65	01000001
E	69	01000101
I	73	01001001
O	79	01001111
U	85	01010101

Introdução à Compressão de Dados

compressão: arquivo original → **codificação** → arquivo compactado



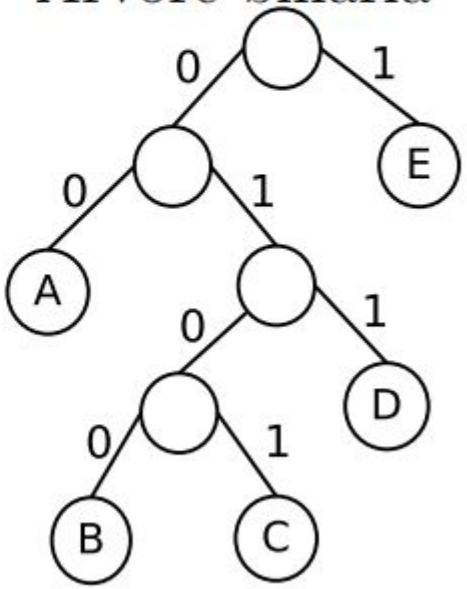
Introdução à Compressão de Dados

- Exemplo: $S = \{A, B, C, D, E\}$

Tabela de símbolos

símbolo	código
A	00
B	0100
C	0101
D	011
E	1

Árvore binária



Bibliografia Básica

- **ANDRÉ BACKES. Estrutura de Dados descomplicada em Linguagem C. Elsevier. (2016).**
- **NÍVIO ZIVIANNE. Projeto de Algoritmos: com Implementações em Java e C++. São Paulo: Cengage Learnin. (2007).**
- **THOMAS H. CORMEN, CHARLES E. LEISERSON, RONALD L. RIVEST e CLIFFORD STEIN. Algoritmos:Teoria e Prática. Elsevier. (2012).**

Estrutura de dados - aulas de implementação

- https://programacaodescomplicada.wordpress.com/indice/estrutura-de-dados/

Curso de Estrutura de Dados

67° Aula: Árvores – Vídeo-Aula

68° Aula: Árvores: propriedades – Vídeo-Aula

69° Aula: Árvore Binária: Definição – Vídeo-Aula

70° Aula: Árvore Binária: Implementação – Vídeo-Aula

71° Aula: Criando e destruindo uma árvore binária – Vídeo-Aula

72° Aula: Árvore Binária: informações básicas – Vídeo-Aula

73° Aula: Percorrendo uma Árvore Binária – Vídeo-Aula

74° Aula: Árvore Binária de Busca – Vídeo-Aula

Avaliação

- Prova 1 (40%)
  - 24/04
- Prova 2 (40%)
  - 19/06
- Participação & Avaliação continuada (20%)
  - Atividades e exercícios de implementação.

Atividades em Sala

1) Considerando os conteúdos de ED1, atribua uma nota (de 0 a 10) sobre o seu conhecimento de cada tópico abaixo e cite, e justifique com base nos conceitos-chave que você se lembra de:
   a) Ponteiros e alocação dinâmica
   b) Listas estáticas/encadeadas/duplamente encadeadas.
   c) Algoritmos de ordenação (selection/quick/merge/lineares).
   d) Análise assintótica e complexidade.
2) O que são tabelas Hash e como elas se relacionam com os conceitos de listas e vetores? Quais as vantagens e desvantagens das tabelas Hash em relação a listas e vetores?
3) Utilizando a noção de Hash, preencha os comentários do código disponibilizado em scanuto.com/hash1.c para 5 alunos cujas matrículas se iniciam em 5538 e terminam em 5542. **Não crie nenhum vetor ou estrutura adicional para armazenar matrículas e notas.**